EVALUATING THE EFFICIENCY OF SEQUENTIAL MINIMAL OPTIMIZATION ALGORITHM IN THE TRAINING OF SUPPORT VECTOR MACHINES

Nguyen Anh Khoa
s1051580
Faculty of Social Sciences
Radboud University

January 9, 2023

ABSTRACT

It is widely known that Support Vector Machine (SVM) is a machine learning technique designed for classification problems. In the development of machine learning and AI, there are numerous methods introduced to train a SVM, one of which is Sequential Minimal Optimization (SMO) proposed by Platt. In this report, we implement SMO algorithm and conduct experiments using practical datasets to study the run time complexity of SVMs trained by this algorithm. Our results show that this algorithm has a low run-time complexity and a great ability to scale with large training datasets, which is in line with the results in Platt's paper. In addition, we take two kernels including linear kernel and Gaussian kernel into consideration to study the time complexity.

1 Introduction

For years, machine learning - a branch of Artificial Intelligence has developed extensively, which leads to the birth of numerous algorithms. In such a long journey of development, such innovative and efficient algorithms have been divided into three main groups including supervised learning, unsupervised learning, and reinforcement learning. Among these types of algorithms, supervised learning is considered as the most traditional and fundamental algorithm, in which Support Vector Machine (SVM) has drawn a remarkable attention from AI researchers in the 2000s [1].

In 1995, Vapnik and his colleagues introduced Support Vector Machine (SVM) as a supervised algorithm which allows the computer to learn and perform tasks involving binary classification [2]. Support Vector Machines (SVM) is a powerful and widely-used supervised learning algorithm that can be applied to a range of tasks including classification, regression, and outlier detection [1]. This algorithm can be considered as a decision support system which utilize a hypothesis space in a higher dimensional space to discover pattern between data, separate and classify them.

While SVM has demonstrated excellent performance on many real-world datasets, one of the challenges in implementing this algorithm is the computation of the optimization problem, which can be computationally expensive for large datasets. In fact, the optimization process in the training of SVM is a quadratic programming (QP) problem [3], which is far more complex than a linear optimization problem. This makes the algorithm become hard to converge when the number of training examples is high, and thus results in a computationally expensive issue.

There have been different attempts to improve the efficiency of SVM training. Some approaches have focused on the use of approximations and heuristics, while others have attempted to address the computational complexity of the optimization problem directly. Vapnik [4] has proposed a method to solve quadratic programming problem in the training of SVM. In general, the idea of his method is to divide a QP problem into smaller problems, each of which is solved individually to find non-zero Lagrange multipliers and ignore the zero multipliers. The algorithm finishes when these problems are completely solved. On the other hand, Ferris et al. [5] used interior point method. It is a type of gradient-based optimization method that utilizes a barrier function to transform the original optimization problem into

a sequence of easier problems that can be solved using Newton's method. However, these algorithms require a large amount of memory and do not fit well with large-scale problems [1], [5].

In this report, we are going to implement Sequential Minimal Optimization (SMO) for training kernel SVM. This algorithm is introduced by John Platt as an effective algorithm to solve quadratic problem in SVM within a reasonable time and memory [1]. Our implementation is highly adapted from Platt's paper. In addition, we conduct experiments on different datasets to study the actual run-time complexity of SMO algorithm.

2 Preliminaries

In this section, we are going to give a brief introduction to SVM and the optimization problem in the training of SVM. We decompose SVM algorithm to different sub-components so that it is easier to understand this algorithm thoroughly. Additionally, we discuss the kernel trick to generalize SVM to deal with non-linearly separable cases.

2.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a classification technique which uses the concept of hyperplane and margin to identify and split data points to two different classes [2], see Figure 1. A hyperplane is a specific plane in a higher dimensional space which is able to separate data points effectively, while a margin is the smallest distance from a data point the hyperplane. Mathematically, hyperplane and margin are represented as:

$$\text{Hyperplane: } P(x) = w \cdot x - b$$

$$\text{Margin: margin} = \min_i \frac{y_i(wx_i - b)}{||w||_2}$$

where w is called a weight vector, and b is called a threshold or a bias

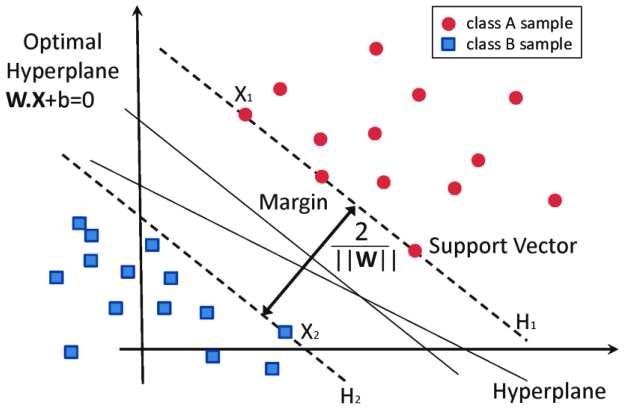

Figure 1: An illustration of hyperplane and margin in SVM [6]

In SVM, the optimization problem is the process of finding w and b such that this margin is maximum [3]:

$$(w,b) = \arg\max_{w,b} \left\{ \min_i \frac{y_i(wx_i - b)}{||w||_2} \right\} = \arg\max_{w,b} \left\{ \frac{1}{||w||_2} \min_i y_i(wx_i - b) \right\}$$

The above optimization problem is difficult to be solved, thus we can transform it into the following problem [1]:

$$(w,b) = \arg\min_{w,b} \frac{1}{2}||w||_2^2 \quad \text{subject to: } y_i(wx_i - b) \geq 1, \forall i = 1,2,\ldots,N \tag{1}$$

Now, by using Lagrange multipliers α_i, we have the Lagrangian problem for (1) [7]:

$$\mathbf{L}(w,b,\alpha) = \frac{1}{2}||w||_2^2 - \sum_{n=1}^{N} \alpha_i \left(y_i(wx_i - b) - 1\right) \tag{2}$$

By taking the partial derivative of (2) with respect to w and b, our optimization problem can be converted into a quadratic programming problem. This is also called Wolfe dual problem and is our main objective in optimizing support vectors in SVM [1]:

$$\alpha = \arg\min_{\alpha} \Psi(\alpha) = \arg\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i x_j - \sum_{i=1}^{N} \alpha_i$$
$$\text{subject to: } 0 \leq \alpha_i \leq C, \forall i = 1, 2, ..., n \tag{3}$$
$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

where C is a constant that helps to balance margin maximization and training error minimization. Once the hyperplane is defined, data points are classified using the following decision function:

$$\mathrm{f}(x_i) = \text{sign}(wx_i - b) = \text{sign}\left(\sum_{j=1}^{N} \alpha_j y_j x_i x_j - b\right) \tag{4}$$

where sign(x) is a sign function which gives 1 if $x \geq 0$ and -1 otherwise. Ultimately, it is guarantee that the quadratic programming problem from (3) is completely solved when Karush-Kuhn-Tucker (KKT) conditions are satisfied:

$$\begin{aligned} \alpha_i = 0 &\iff y_i f(x_i) \geq 1, \\ 0 < \alpha_i < C &\iff y_i f(x_i) = 1, \\ \alpha_i = C &\iff y_i f(x_i) \leq 1 \end{aligned} \tag{5}$$

2.2 Kernel SVM

In practice, the binary classification task is not always straightforward since there are numerous non-linearly separable datasets. Thus, it is important to generalize the SVM algorithm so that it can deal with non-linearly separable cases. This forms the idea of kernel SVM, which is finding a transformation such that the original non-linearly separable data is converted into a linearly separable data in a higher order space. This can be further visualized in Figure 2.

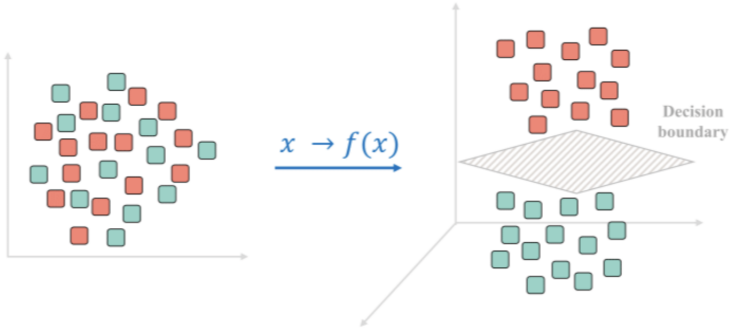

Figure 2: An example of kernel trick [8]. The left image shows a non-linearly separable case in 2D space, while right image shows the transformed data in 3D.

By using the kernel trick, our quadratic programming problem (3) becomes:

$$\alpha = \arg\min_{\alpha} \Psi(\alpha) = \arg\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{K}(x_i, x_j) - \sum_{i=1}^{N} \alpha_i$$
$$\text{subject to: } 0 \leq \alpha_i \leq C, \forall i = 1, 2, ..., n \tag{6}$$
$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

and the decision function (4) becomes:

$$\mathrm{f}(x_i) = \text{sign}\left(\sum_{j=1}^{N} \alpha_j y_j \mathbf{K}(x_i, x_j) - b\right) \tag{7}$$

In this report, we are going to use two popular kernel: linear kernel and gaussian kernel. The mathematical formula for these two kernels are:

$$\text{linear kernel: } \mathbf{K}(x_i, x_j) = x_j \cdot x_j$$
$$\text{gaussian kernel: } \mathbf{K}(x_i, x_j) = \exp(-\gamma ||x_i - x_j||_2^2), \gamma > 0$$

3 Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) is a fast training algorithm for SVM introduced by Platt [1]. In his paper, Platt has verified that SMO works more effectively than other methods [1]. In principle, this algorithm works by selecting two Lagrange multipliers to optimize at each iteration. The algorithm then updates the variables using the gradient of the objective function and a step size. The algorithm repeats this process until the optimal conditions for the QP problem are satisfied, at which point the global solution is found. According to Platt, this algorithm can be solved analytically and is robust against numerical precision issue [1].

3.1 Mathematical formulas

In this section, we present some formulas used in SMO process. For simplicity, we express the kernel computation as $K(x_i, x_j) = K_{ij}$. Since SMO algorithm selects two Lagrange multipliers, i.e α_1 and α_2 with the corresponding training samples (x_1, y_1) and (x_2, y_2), our quadratic optimization problem in (6) is expressed as [9]:

$$\begin{aligned} \alpha = \arg\min_{\alpha_1,\alpha_2} \Psi(\alpha) = \arg\min_{\alpha_1,\alpha_2} \frac{1}{2}\left(\alpha_1^2 K_{11} + 2s\alpha_1\alpha_2 K_{12} + \alpha_2^2 K_{22}\right) \\ \text{subject to: } 0 \le \alpha_i \le C, \forall i = 1,2 \\ s\alpha_1 + \alpha_2 = \gamma \end{aligned} \tag{8}$$

where $s = y_1 y_2$ and $\gamma \in \mathbb{R}$.

In SMO algorithm, two Lagrange multiplier α_1 and α_2 are continously updated through the following procedure [1]:

First of all, we update the second Lagrange multiplier α_2 first. This multiplier can be obtained by calculating:

$$\alpha_2^{\text{new}} = \alpha_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta} \tag{9}$$

where $E_i = f(x_i) - y_i$ is called the error of the output of x_i and $\eta = K_{11} + K_{22} - K_{12}$.

This α_2_new value is then clipped to lie between the constraints:

$$\alpha_2^{\text{new, clipped}} = \begin{cases} L, & \text{if } \alpha_2^{\text{new}} \le L \\ H, & \text{if } \alpha_2^{\text{new}} \ge H \\ \alpha_2^{\text{new}}, & \text{otherwise} \end{cases} \tag{10}$$

where L, H is defined as:

- If $y_1 \neq y_2$:

$$\begin{cases} L &= \max(0, \alpha_2^{\text{old}} - \alpha_1^{\text{old}}) \\ H &= \min(C, C + \alpha_2^{\text{old}} - \alpha_1^{\text{old}}) \end{cases} \tag{11}$$

- If $y_1 = y_2$:

$$\begin{cases} L &= \max(0, \alpha_2^{\text{old}} + \alpha_1^{\text{old}} - C) \\ H &= \min(C, \alpha_2^{\text{old}} + \alpha_1^{\text{old}}) \end{cases} \tag{12}$$

In case $\eta < 0$, $\alpha_2^{\text{new, clipped}}$ is calculated in a different way. Before calculating $\alpha_2^{\text{new, clipped}}$, two additional terms L_{obj} and H_{obj} are computed:

$$\begin{aligned}
f_1 &= y_1(E_1 + b) - \alpha_1^{\text{old}} K_{11} - s\alpha_2^{\text{old}} K_{12} \\
f_2 &= y_2(E_2 + b) - s\alpha_1^{\text{old}} K_{12} - \alpha_2^{\text{old}} K_{22} \\
L_1 &= \alpha_1 + s(\alpha_2 - L) \\
H_1 &= \alpha_1 + s(\alpha_2 - H) \\
L_{\text{obj}} &= L_1(f_1 + f_2) + \frac{1}{2} L_1^2 K_{11} + \frac{1}{2} L^2 K_{22} + sLL_1 K_{12} \\
H_{\text{obj}} &= H_1(f_1 + f_2) + \frac{1}{2} H_1^2 K_{11} + \frac{1}{2} H^2 K_{22} + sHH_1 K_{12}
\end{aligned} \tag{13}$$

Then, these two terms are used to clipped the second multiplier:

$$\alpha_2^{\text{new, clipped}} = \begin{cases} L, & \text{if } L_{\text{obj}} \leq H_{\text{obj}} + \varepsilon \\ H, & \text{if } L_{\text{obj}} \geq H_{\text{obj}} + \varepsilon \\ \alpha_2^{\text{old}}, & \text{otherwise} \end{cases} \tag{14}$$

where ε is an approximation term and is often set to 0.001.

Once we get the new α_2, the first Lagrange multiplier is then updated:

$$\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + s(\alpha_2^{\text{old}} - \alpha_2^{\text{new, clipped}}) \tag{15}$$

Additionally, in the decision function (7), we have the threshold term b. Therefore, this threshold should also be updated during the training process of SMO algorithm:

$$\begin{aligned}
b_1 &= E_1 + y_1(\alpha_1^{\text{new}} - \alpha_1^{old})K_{11} + y_2(\alpha_2^{\text{new, clipped}} - \alpha_2^{old})K_{12} + b \\
b_2 &= E_2 + y_1(\alpha_1^{\text{new}} - \alpha_1^{old})K_{12} + y_2(\alpha_2^{\text{new, clipped}} - \alpha_2^{old})K_{22} + b \\
b^{\text{new}} &= \frac{b_1 + b_2}{2}
\end{aligned} \tag{16}$$

3.2 Implementation of SMO

In this section, we are going to present the details of how SMO works through pieces of pseucode. Our pseudocode is simply an adaptation from the ones proposed by Platt in his paper[1]. Our full implementation can be found HERE. In the training of SMO algorithm, there are some parameters needed to be initialized so that the algorithm can function properly:

- Lagrange multipliers list **alpha**: this list is initialized to 0 and has the length equals to the total number of training examples.
- Threshold term **b**: this is also initialized to 0.
- Error cache **error_cache**: this is a list which stores all errors in the training of SMO. It is initialized to 0, and then updated for the first time using $E_i = -y_i$
- Weight vector **w**: this vector is only used when linear kernel is used. In this case, w is initialized to a list of 0

Algorithm 1 is the main loop of SMO algorithm. First of all, it initializes all necessary parameters described above. Then, this algorithm iterates through every training examples to check whether the corresponding Lagrange multiplier α_i violates the KKT conditions in equation (5). If it violates the conditions, the procedure examine_example() is called to examine the second Lagrange multiplier, which is used to optimize Lagrange multipliers. The algorithm converges when all multipliers are optimized.

Algorithm 2 shows how SMO algorithm find an example that can be used to improve the SVM classifier by identifying a pair of examples that violate the optimality conditions for the current solution. If such a pair is found, this algorithm updates the SVM classifier by finding the optimal values for the Lagrange multipliers (alpha) associated with these examples and updating the bias term using take_step() procedure. For a defined first Lagrange multiplier i, the algorithm choose a second multiplier j. In Platt's paper, the second multiplier is chosen such that difference $|E_i - E_j|$ is maximum, which is demonstrated in algorithm 3.

Algorithm 1: fit()

```
Initialize α, b and error_cache;
if linear kernel then
    initialize w;
end
examine_all ← True;
num_changed ← 0;
while num_changed > 0 || examine_all do
    num_changed = 0;
    if examine_all then
        forall i in the training examples do
            if not is_satisfied_KKT(i) then
                num_changed ← num_changed + examine_example(i);
            end
        end
        examine_all ← False;
    else
        forall i in the training examples such that 0 < α[i] < C do
            if not is_satisfied_KKT(i) then
                num_changed ← num_changed + examine_example(i);
            end
        end
        examine_all ← True;
    end
end
```

Algorithm 2: examine_example(i)

```
if exist j ≠ i and 0 < α[j] < C then
    j ← choose_second_alpha(i);
    if take_step(j, i) then
        return 1;
    end
end
forall remaining examples such that 0 < α_i < C do
    pick j randomly;
    if take_step(j, i) then
        return 1 ;
    end
end
forall possible training examples do
    pick j randomly;
    if take_step(j, i) then
        return 1;
    end
end
```

For the last step which is algorithm 4, a method take_step() is implemented. In a nutshell, this procedure calculates and updates new value for both Lagrange multipliers α_i and α_j. It consists of multiple pieces of code that compute α_2^{new}, clip α_2^{new}, compute α_1^{new}, update threshold b, update error cache at index i, j ,and store new multipliers. All these sub-steps can be implemented straightforwardly using formulas described in section 3.1

Algorithm 3: choose_second_alpha(i)

error_i ← error_cache[i];
abs_values ← [];
forall *error e in error_cache* **do**
    abs ← e - error_i;
    Add abs to abs_values;
end
j ← the position of the maximum element in abs_values;
return j;

Algorithm 4: take_step(i1, i2)

if *i1 = i2* **then**
    return 0;
end
$\alpha_1^{\text{old}} \leftarrow \alpha[i1]$;
$\alpha_2^{\text{old}} \leftarrow \alpha[i2]$;
s ← y[i1] · y[i2];
L, H ← compute_L_H(i1, i2) using equation 11 and 12;
$\eta \leftarrow K_{11} + K_{22} - K_{12}$;
if $\eta > 0$ **then**
    $\alpha_2^{\text{new}} \leftarrow$ update α_2^{new} using equation 9;
    $\alpha_2^{\text{new}} \leftarrow$ clip α_2^{new} using equation 10;
end
else
    $L_{\text{obj}}, H_{\text{obj}} \leftarrow$ compute L_obj and H_obj through 13;
    $\alpha_2^{\text{new}} \leftarrow$ clip α_2^{old} using 14;
end
if $|\alpha_2^{new} - \alpha_2^{old}| < \varepsilon \cdot (\alpha_2^{new} + \alpha_2^{old} + \varepsilon)$ **then**
    return 0;
end
$\alpha_1^{\text{new}} \leftarrow$ compute α_1^{new} using equation 15;
update threshold b using equation 16;
if *linear kernel* **then**
    $w \leftarrow w + y_1(\alpha_1^{\text{new}} - \alpha_1)x_1 + y_2(\alpha_2^{\text{new}} - \alpha_2)x_2$;
end
update error_cache by computing $E_i = f(x_i) - y_i$ where $f(x_i)$ is calculated using equation 7;
$\alpha[i1] \leftarrow \alpha_1^{\text{new}}$;
$\alpha[i2] \leftarrow \alpha_2^{\text{new}}$;
return 1;

4 Experiments and Results

4.1 Experimental setup

In this report, we make use of Adult dataset [10] available in UCI repository to experiment our implementation of SMO algorithm and study the run-time complexity of this algorithm. This is a widely known dataset for research in machine learning and used to evaluate the performance of various classification algorithms. It consists of a total of 48842 instances and 14 attributes. The goal of the dataset is to predict whether an individual's income is above or below $50,000 per year based on the given attributes.

During the exploratory data analysis process, we realize that there are some missing values in this dataset. We decide to replace these missing values by the value which has the highest occurance frequency in the same attribute. In addition, all categorical features including workclass, education, marital-status, occupation, relationship, race, sex, native-country are encoded into numerical values. Since SVM utilizes a sign function to classify data points, we also encode the dependent variable which is income to 1 and -1. If income is "$\leq 50K$", it is encoded to -1, otherwise it is encoded to 1.

As mentioned in the beginning of this report, we are going to study the run-time complexity of SMO algorithm. Since SMO complexity depends on different parameters such as training size, the choice of C, etc, we only focus on studying the complexity as a function of training size. Therefore, we fragment the Adult dataset into 10 sub-datasets of different size described in Table 1. These smaller datasets are then used to train SVM with SMO algorithm and the execution time is recorded. In our experiments, both linear and gaussian kernel are used, constant C is set to 1.0, and ε term is set to 0.001.

	Size	
Name	Training set	Testing set
Subset 1	(326, 14)	(326,)
Subset 2	(651, 14)	(651,)
Subset 3	(977, 14)	(977,)
Subset 4	(1302, 14)	(1302,)
Subset 5	(1628, 14)	(1628,)
Subset 6	(1954, 14)	(1954,)
Subset 7	(2279, 14)	(2279,)
Subset 8	(2605, 14)	(2605,)
Subset 9	(2930, 14)	(2930,)
Subset 10	(3256, 14)	(3256,)

Table 1: Information about size of each subset. Each dataset has a training set and a testing set. The size of each set is displayed in the format (a,b) where a is the number of examples, and b is the number of attributes in the dataset

4.2 Results

In this section, we are going to present the obtained results from the experiment process. We measure both execution time and the accuracy score for each dataset. Since there is a randomization in SMO algorithm, the results can be different each time. However, by conducting several experiments, we have confirmed that the general pattern of the results do not change.

Subset	Linear kernel	Gaussian kernel
Subset 1	0.575	0.727
Subset 2	0.557	0.763
Subset 3	0.694	0.699
Subset 4	0.778	0.770
Subset 5	0.521	0.745
Subset 6	0.549	0.765
Subset 7	0.612	0.739
Subset 8	0.647	0.800
Subset 9	0.522	0.768
Subset 10	0.438	0.776

(a) Accuracy for each subset

Subset	Linear kernel	Gaussian kernel
Subset 1	0.194	1.829
Subset 2	0.925	7.179
Subset 3	4.030	15.841
Subset 4	6.461	27.686
Subset 5	11.396	42.933
Subset 6	19.365	61.358
Subset 7	24.178	85.363
Subset 8	30.507	116.450
Subset 9	41.726	150.609
Subset 10	48.510	199.881

(b) Training time of each subset in seconds (s)

Table 2: Results of the training of SMO algorithm. The left table shows the training time for each dataset, while the right table shows the accuracy of this algorithm for each dataset.

Table 2a provides an insight into the performance of SMO algorithm, i.e the accuracy of the algorithm. Overall, it can be observed that a Gaussian kernel SMO tends to yield a better accuracy than a Linear kernel one. In specific, the accuracy of Linear kernel changes more significantly depending on the dataset than the Gaussian kernel. For instance, the accuracy for Linear kernel SMO varies between 0.40 and 0.65, while the Gaussian kernel SMO is approximately stable between 0.70 and 0.80. This pattern is consistent to the fact that for such a non-linearly separable dataset (Adult dataset), a SVM with Gaussian kernel performs better than Linear kernel.

Table 2b presents the real-time duration to completely train a SVM using SMO algorithm. Generally, both kernels training time scales remarkably with respect to the number of training examples. Although a linear kernel SVM gives a

worse accuracy than the Gaussian kernel, it saves more time in the training process. This is indeed a trade-offs between complexity and classification error in SVM as described in [11]. By having a look at Figure 3, we can find out how the run-time complexity of Linear SVM and Gaussian SVM depends on the total training examples. This figure confirms that a SVM with Linear kernel has the time complexity of $O(n^2)$, meanwhile Gaussian kernel SVM's time complexity is $O(n^3)$ where n is the number of training examples [12].

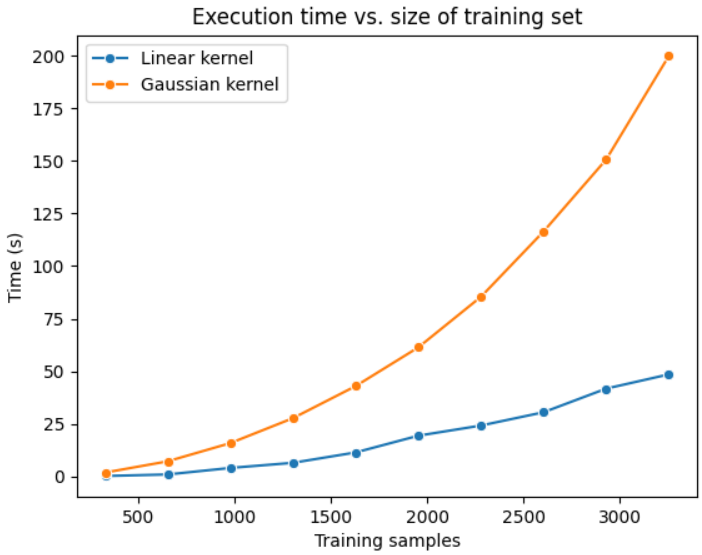

Figure 3: Training time of SVM using SMO method (on y-axis) as a function of number of training samples (on x-axis). The Linear SVM is colored in orange, and the Gaussian kernel is colored in blue.

5 Discussion and conclusion

There are two main aims of this project. The initial goal is to implement SMO algorithm introduced by Platt [1]. The second goal, which is the most important goal of the project, is to conduct experiment using a practical dataset to study the run-time complexity of this algorithm. These two aims have been documented in details throughout this report.

SMO has proved its efficiency in solving the quadratic programming problem (3) which arises in the training of SVM by breaking down the large problem into sub-problems and finding the pair of Lagrange multipliers for these smaller problems. The obtained results meet our expectation and show such an efficiency. Although our implementation does not result in a good performance as the standard SVM library provided by sklearn, it is still acceptable as the general pattern is consistent to other research papers.

These days, SMO algorithm is still considered as the fastest algorithm to train SVM compared to other methods. Despite the fact there are various modern techniques to train the computer for a specific classification task, SVM remains useful due to its generality, run-time efficiency, and high applicability [13].

References

[1] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[2] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995. ISBN 0-387-94559-8.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[4] Vladimir Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[5] Michael C Ferris and Todd S Munson. Interior-point methods for massive support vector machines. *SIAM Journal on Optimization*, 13(3):783–804, 2002.

[6] Esperanza García-Gonzalo, Zulima Fernández-Muñiz, Paulino Jose Garcia Nieto, Antonio Sánchez, and Marta Menéndez. Hard-rock stability analysis for span design in entry-type excavations with learning classifiers. *Materials*, 9:531, 06 2016. doi:10.3390/ma9070531.

[7] Tong Wen and Alan Edelman. Support vector machine lagrange multipliers and simplex volume decompositions. *submitted to SIAM journal on Matrix Analysis and Applications*, 2000.

[8] Mohammadreza Sheykhmousa and Masoud Mahdianpari. Support vector machine vs. random forest for remote sensing image classification: A meta-analysis and systematic review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10 2020. doi:10.1109/JSTARS.2020.3026724.

[9] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[10] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[11] Haldun Aytug and Serpil Sayın. Exploring the trade-off between generalization and empirical errors in a one-norm svm. *European journal of operational research*, 218(3):667–675, 2012.

[12] Abdiansah Abdiansah and Retantyo Wardoyo. Time complexity analysis of support vector machines (svm) in libsvm. *International journal computer and application*, 128(3):28–34, 2015.

[13] Jing Yi Tou, Yong Haur Tay, and Phooi Yee Lau. Recent trends in texture classification: a review. In *Symposium on Progress in Information & Communication Technology*, volume 3, pages 56–59, 2009.